MISB MOTION IMAGERY STANDARDS BOARD	MISB ST 1303.2
STANDARD **Multi-Dimensional Array Pack**	**25 June 2020**

1 Scope

This standard describes a method for formatting multi-dimensional arrays of data in KLV (Key Length Value). Multi-dimensional arrays store and organize related data. Applications may process the array as a unit of data (i.e., matrices, etc.) or use them to organize a group of information. This standard defines a KLV Pack construct to format a multi-dimensional array and the array support information. The Multi-Dimensional Array Pack (MDAP) minimizes the bytes needed to represent the data. Supporting information includes options for specifying methods of array packing and element processing, such as using MISB ST 1201 (Floating Point to Integer Mapping) for compression and Run-Length Encoding for unsigned integers.

In application, the Multi-Dimensional Array Pack defined in this standard requires further context from an invoking document; that is, it is not a standalone construct. To provide consistency, this standard defines a syntax for invoking this standard within MISB documents.

2 References

[1] SMPTE ST 336:2017 Data Encoding Protocol Using Key-Length-Value.

[2] MISB, MISP-2020.1: Motion Imagery Handbook, MISB, 2020.

[3] MISB ST 1201.4 Floating Point to Integer Mapping, Feb 2019.

[4] ISO/IEC 8825-1:2015 (ITU-T X.690) Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canocical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

[5] IEEE 754-2008 Standard for Floating-Point Arithmetic.

3 Revision History

Revision	Date	Summary of Changes

ST 1303.2	06/25/2020	• Added array compression techniques:
		  ○ Boolean Array Encoding
		  ○ Unsigned Integer Encoding
		  ○ Run-Length Encoding
		• General Editing

		• Broadened the Element Processing concept; renamed it to Array Processing
		  ○ Element Processing is now one of the formatting options
		  ○ Renamed EPA and EPAS acronyms to APA and APAS respectively
		• MDARRAY Syntax Changes:
		  ○ "MDARRAY" changed to "MDAP" since syntax changes
		  ○ Removed use of KLV Keys
		  ○ Added Type to declare data type of array elements
		• Updated Examples with new MDAP syntax
		• Added requirements -20, 21
		• Deprecated requirements: -1, 4, 5, 6, 8, 10, 11, 12, 13, 15, 18

4 Acronyms, Terms, Definitions

APA	Array Processing Algorithm
APAS	Array Processing Algorithm Support
IMAPA	Integer Mapping Starting Point A; defined in MISB ST 1201
IMAPB	Integer Mapping Starting Point B; defined in MISB ST 1201
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
KLV	Key Length Value
MDAP	Multi-Dimensional Array Pack
MISB	Motion Imagery Standards Board
MISP	Motion Imagery Standards Profile
SMPTE	Society of Motion Pictures & Television Engineers
ST	Standard

5 Introduction

Within software, data structures such as records and arrays provide information organization. When reformatting data structures into KLV, record structures map readily into KLV constructs, such as KLV Set and Pack constructs. However, the SMPTE KLV standard [1] does not formally specify a means for reformatting arrays into KLV. This standard defines an efficient and flexible method for formatting multi-dimensional arrays into KLV.

6 Array Composition

An array of data is an organization of multiple elements (i.e., the data) along one or more dimensions. In transferring an array of data, support information ensures the organization of the data is consistent between the sender and receiver. A document which invokes this standard will define the array parameters to include: the number of dimensions, the number of elements per dimension, the size of an element, and the data's type.

The elements in an array can be of any type (e.g., integers, floating point values, strings, KLV pack or set values, etc.). For the MDAP the size of the elements is fixed-length but some MDAP options allow for element lengths to be computable from their binary format (e.g., BER-OID). Inflating or shrinking elements can normalize elements with different sizes into fixed-length elements (see Appendix B).

When grouping data into an array information about the group can help reduce the number of bytes needed for the array. For example, in an array composed of integers originally defined with four bytes each, but the values only need one byte, the array uses one-byte elements instead of four.

This document uses the following terminology:

Array: Uppercase notation to indicate reference to an array as defined in this document.

Element: Data representing a value identified by an index or indices.

Requirement(s)	
ST 1303-02	The invoking standard shall specify the number of dimensions of the Array.
ST 1303-03	The invoking standard shall specify the size of each dimension or how to compute the size of each dimension in the Array.

Figure 1 illustrates a single dimension Array, where its indexed Elements begin at 0 and end at N-1 for N-Element columns. Elements in a one-dimensional array index only by columns, so column 0 is Element 0, column 1 is Element 1, etc.

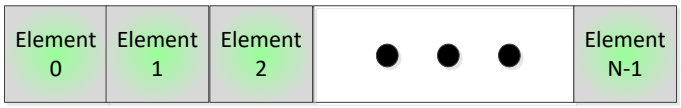

Figure 1: Illustration of a One-Dimensional Array of N Elements.

Figure 2 shows a two-dimensional Array, where its Elements index by the row r and the column c. Thus, the first Element is (0, 0) for r = 0, c = 0; the second Element in row 0 is (0, 1) for r = 0, c = 1; the second Element in row 1 is (1, 1) for r = 1 and c = 1, etc. (see Figure 2). Rows index 0 to M-1 and columns index 0 to N-1.

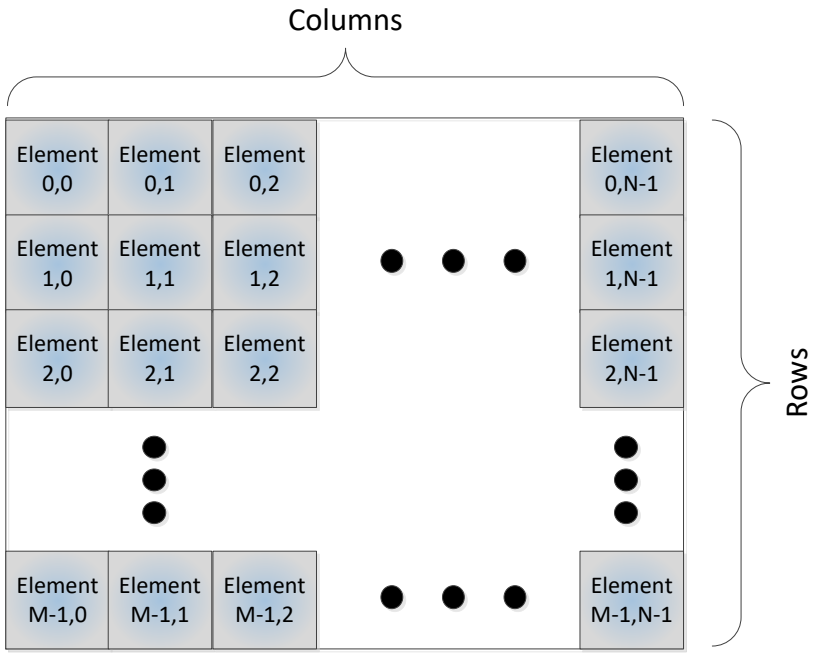

Figure 2: Illustration of a Two-Dimensional Array of M-Rows by N-Columns

Figure 3 depicts a three-dimensional Array, where its Elements index by planes (shown as a separate colored two-dimensional array), rows, and columns. The first Element in plane 0, row 0, and column 0 is (0, 0, 0) for (p, r, c) where p = r = c = 0. Planes index 0 to P-1, rows index 0 to M-1, and columns index 0 to N-1.

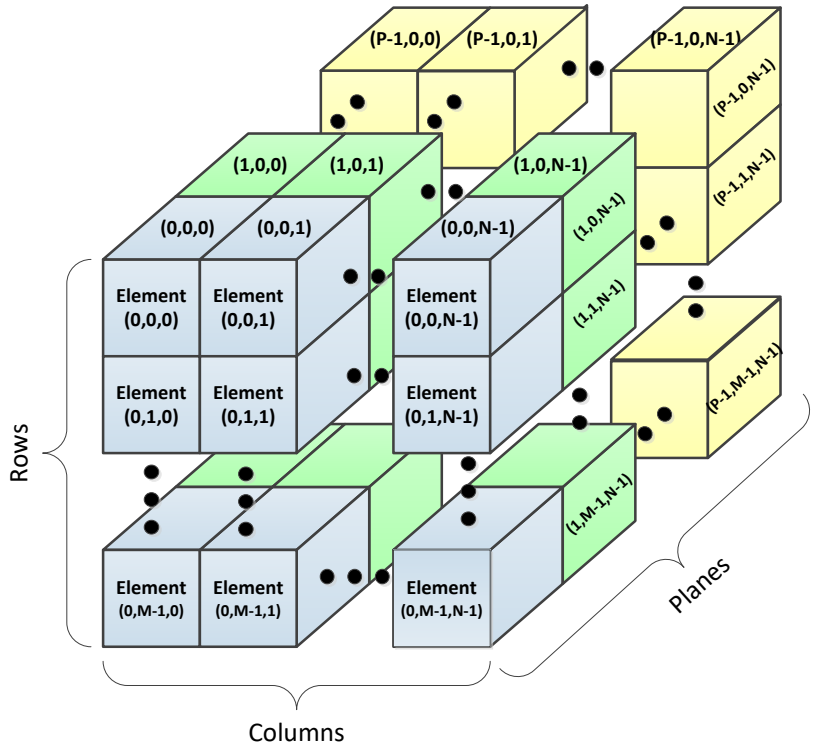

Figure 3: Illustration of a Three-Dimensional Array of P-Planes by M-Rows by N-Columns

Higher number of dimensions extend the notation on the left-hand side. For example, four dimensional arrays include the fourth dimension on the left of the "plane" dimension, e.g., (d_4, plane, row, column) and five dimensional arrays include the fifth dimension to the left of the 4th dimension, e.g., (d_5, d_4, plane, row, column), etc.

7 Multi-Dimensional Array Pack

The Multi-Dimensional Array Pack (MDAP) is a KLV Truncation Pack (described in the Motion Imagery Handbook [2]) which contains both the array data and information about the array. The array data has three formatting types: **Natural**, **Element-Processed**, and **Compact**. The **Natural** format arranges the multi-dimensional array into a serialized one-dimensional array. The serialization for the Natural format uses same length array Elements, i.e., all Elements in MDAP are the same number of bytes. For example, a Natural formatted 3 x 3 array of floating-point values becomes a one-dimensional array with nine Elements; each Element is the same length floating point, e.g., a 32-bit IEEE. The MDAP includes the number of dimensions, size of each dimension, and size of each Element so data receivers can re-construct the array.

An **Element-Processed** array is like a Natural formatted array, but each Element is pre-processed before the serialization. For example, a 3x3 array of floating-point values may be Element-Processed with MISB ST 1201's [3] IMAP encoding (see Appendix D.1) before the serialization. The IMAP algorithm reduces the size of the Elements. All Elements of the array undergo the same IMAP processing, thus sharing the same minimum, maximum, and length

parameters and reducing the size of each element to the same number of bytes. The MDAP includes the IMAP parameters so data receivers can decode the array elements.

Requirement(s)	
ST 1303.2-20	All Elements in a Natural Array shall use the same number of bytes.
ST 1303.2-21	All Elements in an Element-Processed Array shall use the same number of bytes.

A **Compact** array condenses the MDAP array data based on the data type or other means (such as patterns within the array data). The MDAP includes algorithm parameters to support the method of compaction.

To distinguish between the different formatting options and methods the MDAP includes an Array Processing Algorithm (APA) indicator; Appendix D lists and provides details of all APAs.

Table 1 defines the order of the Multi-Dimensional Array Pack items.

Table 1: Multi-Dimensional Array Pack

Name	Required/ Optional	Type	Min Value	Section	Description
N_{Dim}	Required	BER-OID	1	7.2.1	Number of dimensions in the Array. The minimum value is one.
Dim_i	Required	BER-OID	1	7.2.2	Size of i^{th} dimension. There are N_{Dim} number of these values.
E_{Bytes}	Required	BER-OID	0	7.2.3	Number of bytes for each Element. The minimum value is zero.
APA	Required	BER-OID	1	7.2.4	Array Processing Algorithm. Indicator of the format the Array of Elements uses.
APAS	Optional	Defined by APA	N/A	7.2.5	Array Processing Algorithm Support
Array of Elements	Optional	Defined by Invoking Document	N/A	7.2.6	Array data in either Natural, Element-Processed, or Compact format

Figure 4 illustrates the Multi-Dimensional Array Pack, along with its Length. The items (blue) are the required information stated in Table 1; APAS items (yellow) are optional; the Array of Elements (green) constitutes the Array data, which is also optional.

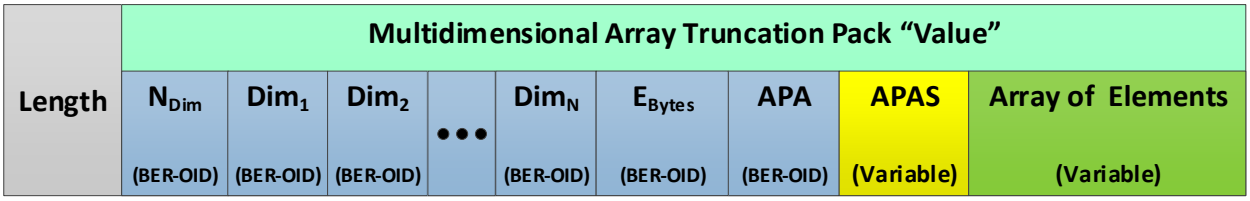

Figure 4: Multi-Dimensional Pack Structure

Depending on the APA value, the Array of Elements may be in a Natural, Element-Processed, or Compact format. If applications choose one of the Compact formats the size of each dimension, Dim_i, is the size before compaction processing of the Array of Elements.

7.1 Multi-Dimensional Array Pack – Length

The Length of the Multi-Dimensional Array Pack is BER-length encoded as specified in [1]. The length includes the number of bytes for N_{Dim}, all the dimensions ($Dim_1 \ldots Dim_N$), the number of bytes for each element E_{Bytes}, the APA, the APAS, and the Array of Elements. Compute $Pack_{Length}$ as follows:

$Pack_{Length} = L(N_{Dim}) + L(Dim_1) + \ldots + L(Dim_N) + L(E_{Bytes}) + L(APA) + L(APAS) + L(Array\ of\ Elements)$

Where L(**x**) is the length of value **x** in bytes.

As discussed below, the L(APAS) and L(Array of Elements) can both be zero depending on usage and circumstances. Appendix D defines the APAS lengths or how to compute them for each APA.

7.2 Multi-Dimensional Array Pack – Value

The Value of the Multi-Dimensional Array Pack consists of a collection of required and optional items as listed in the following sections. The pack requires all items unless noted as optional.

7.2.1 Item - N_{Dim}

The N_{Dim} item is the count of dimensions in the Array of Elements. For example, a simple list of Elements is a one-dimensional Array; therefore, N_{Dim} equals 1. A rectangle of Elements is a two-dimensional Array; therefore, N_{Dim} equals 2. A cube of elements is a three-dimensional Array; therefore, N_{Dim} equals 3, etc. The N_{Dim} value cannot be zero.

Requirement	
ST 1303-07	The invoking document shall specify a value for N_{Dim} that is greater than or equal to one (1).

7.2.2 Item – Dim_i

The Dim_i item specifies the number of Elements in the i^{th} dimension, which is greater than or equal to one (i.e., zero is disallowed). For example, for a three-dimensional cube Array of 100x200x300 Elements (i.e., $N_{Dim} = 3$) $Dim_1 = 100$, $Dim_2 = 200$ and $Dim_3 = 300$.

Requirement	
ST 1303-09	The value of Dim_i shall be greater than or equal to one (1).

7.2.3 Item – E_{Bytes}

Depending on the Array Processing Algorithm (APA) the E_{Bytes} value represents either the Element size (in bytes) or is the value one (1) for arrays which have variable length elements.

With Natural (i.e., APA = 1) or Element-Processed Arrays (i.e., APA=2) the E_{Bytes} item specifies the number of bytes for each fixed-length Element within the Array. For example, if the Array consists of single-precision, floating-point numbers, then E_{Bytes} = 4 bytes.

With Compact Arrays the E_{Bytes} value may indicate a size for use in the APA algorithm or be the value of one for APA algorithms using variable length Element sizes – the value of one is a placeholder and has no meaning in these cases.

If E_{Bytes} is zero the Array of Elements item contains no data and the empty array is just a signal; for example, indicating there was no change in the last set of measurements.

7.2.4 Item – APA (Array Processing Algorithm)

The APA (Array Processing Algorithm) item either specifies the method of processing individual Elements (Element-Processed) or how to decode the Array Element data into an Array (Compact). An APA of one (1) indicates no processing (i.e., Natural Array), and the invoking documents specifies the Element's data type in the Array. All other APA values indicate some form of Element or Array processing. Appendix D lists the different Element Processing Algorithms, their item, and algorithms.

Requirement	
ST 1303-14	An APA (Array Processing Algorithm) item value shall be assigned a value only from MISB ST 1303 Table 3: Array Processing Algorithms.

7.2.5 Item – APAS (Array Processing Algorithm Support) (Optional)

The optional APAS (Array Processing Algorithm Support) item specifies one or more values for the processing algorithm as needed. Each APA in Appendix D lists its support values. If the APA value is set to one (1), there is no corresponding APAS item. The APAS item precedes the Array so parsers can use the information to interpret the data on input.

7.2.6 Item – Array of Elements (Optional)

The optional Array of Elements item has threes formats: Natural, Element-Processed, or Compact. The value of the APA determines the type of format; see Table 3 in Appendix D.

7.2.6.1 Natural and Element-Processed Formats

Both the Natural and Element-Processed formats serialize the multi-dimensional array into a one-dimensional array using row major order. The one-dimensional array contains $Dim_1 * Dim_2 * \ldots * Dim_N$ Elements. The size of each Element is exactly E_{Bytes} bytes. The total length (in bytes) of the Array of Elements is: $Dim_1 * Dim_2 * \ldots * Dim_N * E_{Bytes}$. Any multi-dimensional Element in its serialized one-dimensional Array may be addressed by computing an offset from the start of the Array (see Appendix B for computation).

7.2.6.2 Compact Format

With the Compact format, the Array of Elements is an encoding of the multi-dimensional array. The APA item specifies the encoding method and the optional APAS item may include one or

more parameters to aid in the decoding of the Array of Elements data. The sub-appendices of Appendix D provide details of each encoding method.

8 MISP and MISB Document Standard Notation

When using or "invoking" the MDAP it is important to be clear as to the type, size and optional packing used. The following notation will ensure consistency and completeness of the Array definition:

MDAP(<Type>, <Data Identifier>, <N_{Dims}>, <Dim_1>, …, <Dim_N>, [E_{Bytes}])[1]

Where:

Parameters in <..> are required
Parameters in [..] are optional
<Type> defines the data format (e.g., Float, UInt, IMAP,etc.) of all Elements in the array (see Section 8.1)
<Data Identifier> states the meaning of the value (see Section 8.1)
Section 7 defines other parameters

There are cases when some of the values in the notation will be unknown (i.e., values computed at runtime); the invoking document will label these values with a footnote (i.e., "$Note_x$") which references text providing further explanation. Footnotes can indicate multiple parameters and multiple uses of the MDAP definition - see Examples in Section 8.2. E_{Bytes} can be determined at runtime based on the data values in the Array; however, an invoking standard may specify a predetermined size. Although E_{Bytes} is a mandatory parameter in the MDAP, the Standard Notation lists it as "optional" because the E_{bytes} value may be determined at run time.

Requirement	
ST 1303-16	When using MISB ST 1303, the invoking standard shall use the MDAP(…) notation to ensure consistency and completeness of the Multi-Dimensional Array Pack definition in accordance with MISB ST 1303.

[1] Previous versions of ST1303 use the KLV dictionary item to specify type and data identifier. If an invoking document uses a key instead of a Type, please refer to ST1303.1 for how to interpret the type and meaning.

8.1 Type and Data Identifier

The Type states the format of all Elements in the array. Types include Floating Point, IMAP, Integer, UInt, BER-OID, and others the invoking standard may define. The type is either a fixed length value or the length is determinable from the Element's context (e.g., BER-OID).

The Data Identifier specifies what the Array of Elements represent. Data Identifiers can be associated to the whole Array (homogeneous), or to parts of the Array (heterogeneous) when the Elements all share the same length in bytes. A heterogeneous array example is a grid of points with values representing latitude, longitude and height above ellipsoid (HAE); a three-dimensional Array of points with a first plane used for latitude values, a second plane used for longitude values and a third plane for HAE values as illustrated in Figure 5. This example assumes the latitude, longitude and HAE values use the same number of bytes and Type, e.g., all single-precision, floating-point values.

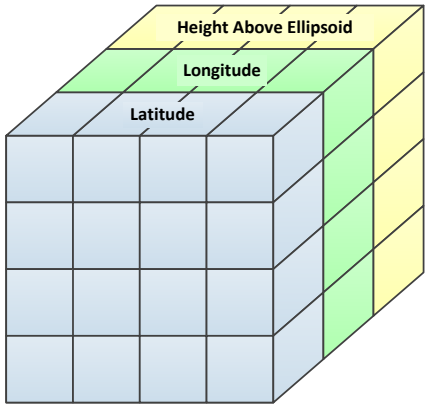

Figure 5: Illustration of 3-D Array with Different Plane Types

When the Array is heterogeneous additional footnotes describe the meaning and usage of the dimensions. For example, in Figure 5, Array(0, all rows, all columns) are Latitude items; Array(1, all rows, all columns) are Longitude items, and Array(2, all rows, all columns) are HAE items. Although the meaning for the dimension may be heterogeneous the Type is the same across the whole array.

When the Array is heterogeneous and the Elements use an Element-Processing APA, the Element values need to be compatible with the APA. Care in choosing the APAS parameters is important because the data may have different value ranges.

Example 1 and 2 below pertain to homogeneous Data Identifiers Arrays, while Examples 3, 4, and 5 show three different approaches to defining a heterogeneous Array.

8.2 Examples

Example 1: An Array of homogeneous Range Depth data.

MDAP(Float, Range Depth, 2, 100, 100)

This is a two-dimensional Array of 100x100 Floating Point Elements of Range Depth data. Note: the use of 32-bit vice 64-bit floating point values is determined at run-time.

Example 2: An Array of homogeneous Range Depth data with footnotes describing the "run-time" values.

MDAP($Note_A$, Range Depth, 2, $Note_B$, $Note_B$)

$Note_A$: Applications determine the type of "Float" or "IMAPB" at runtime
$Note_B$: This value is dependent on the dimensions of the sensor.
Implementations should use APA 2, where Min and Max are the bounds of the Array data, and the Element size (E_{Bytes}) for the IMAPB is determined by IMAPA(Min, Max, 1.0e-4).

This is a two-dimensional Array of Range Depth data from a sensor (dimensions known only at runtime - $Note_B$). The invoking standard recommends the use of IMAPB by defining an APA transformation based on the expected values in the Array. In this example, the recommendation is to use APA two (2), so each Element's value maps to an integer with IMAPB by using the min/max of the source array data and E_{Bytes}. The Element Size (E_{Bytes}) is computed based on the IMAPA computation for length using a precision value of 1.0e-4. Applications determine the Element type of "Float" (APA=1) or "IMAPB" (APA=2) at run-time.

Example 3: An Array of heterogeneous Latitude/Longitude/Altitude data with footnotes describing the dimensional use and "run-time" values.

MDAP(Float, $Note_A$, 3, 3, $Note_B$, $Note_B$)

$Note_A$: Array(0, r, c) = Latitude,
Array(1, r, c) = Longitude,
Array(2, r, c) = HAE,
r = all rows and c = all columns.

$Note_B$: This value is dependent on the dimensions of the sensor.

This is a three-dimensional Array of Latitude, Longitude and Altitude data from a sensor (dimensions known only at runtime - $Note_B$). The first plane (plane 0) is the Latitude, the second plane (plane 1) is the Longitude and the third plane (plane 2) is the Altitude.

This usage is not recommended when applying APA=2 (MISB ST 1201 Floating Point to Integer Mapping) to reduce the Element size (E_{Bytes}) because it is impeded by the need to represent values with ranges orders of magnitude apart using a single mapping (latitudes of +/-90° versus HAE of -900 to 19000 meters).

Example 4: The same data of Example 3 but defined as three separate homogeneous Arrays.

MDAP($Note_A$, Latitude, 2, $Note_B$, $Note_B$)

MDAP($Note_A$, Longitude, 2, $Note_B$, $Note_B$)

MDAP($Note_A$, HAE, 2, $Note_B$, $Note_B$)

$Note_A$: Applications determine the type of Float or IMAPB at runtime

$Note_B$: This value is dependent on the size of the sensor.

These are three separate Arrays: the first Array contains Latitude values; the second Array contains Longitude values; the third Array contains Altitude values. The dimension of each Array is defined at runtime from a sensor's information (dimensions known only at runtime - $Note_A$).

This usage is preferable when applying APA=2 (MISB ST 1201 Floating Point to Integer Mapping) to reduce the Element size (E_{Bytes}) as each Array of Elements transforms individually.

Example 5: An ARRAY using the Location Truncation Pack.

MDAP(Pack, Location Pack, 2, rows, cols)

This is a two-dimensional Array of Location Truncation Pack values. The invoking standard defines the format of the fixed length Location Truncation Pack, with each value potentially defining sub-components: latitude, longitude, height, sigma latitude, sigma longitude, sigma height, Rho lat/lon, Rho lat/height and Rho lon/height. The Element size (E_{Bytes}) determines which data is included in each Elements pack - the Elements must be consistently defined to maintain the required fixed Element size.

This usage rigidly defines the ranges of each sub-component based on the pack definition, so adjusting the Element size (E_{Bytes}) based on the range of data within the Array cannot be used.

Appendix A Deprecated Requirements

Requirement(s)	
ST 1303-01 (Deprecated)	All elements in an Array shall use the same number of bytes.
ST 1303-04 (Deprecated)	The invoking standard shall specify the data contained in the Array using KLV Keys or KLV symbols.
ST 1303-05 (Deprecated)	A Multi-Dimensional Array Pack shall have its mandatory parameters N_{DIM}, Dim_i, E_{Bytes}, APA and its optional elements ordered as defined in MISB ST 1303 Table 1: Multi-Dimensional Array Pack.
ST 1303-06 (Deprecated)	The 16-byte Key for the Multidimensional Array Truncation Pack shall be 06.0E.2B.34.02.05.01.01.0E.01.03.03.06.00.00.00 (CRC 39697).
ST 1303-08 (Deprecated)	The encoding of N_{Dim} shall be BER-OID [4].
ST 1303-10 (Deprecated)	The encoding of each Dim_i shall be BER-OID [4].
ST 1303-11 (Deprecated)	The encoding of E_{Bytes} shall be BER-OID [4].
ST 1303-12 (Deprecated)	The value of E_{Bytes} shall be greater than or equal to zero (0).
ST 1303-13 (Deprecated)	E_{Bytes} equal to zero (0) shall signal that the Array of Elements is not included in the Multi-Dimensional Array Pack.
ST 1303-15 (Deprecated	The data in the Array of Elements shall be organized in row major order.
ST 1303-18 (Deprecated	If EPA is 0x02, then the Array Elements shall be encoded as IMAPB(Minimum, Maximum, E_{Bytes}).

Appendix B Row Major Computation

The following equations demonstrate how to compute the offsets into an Array based on the Array dimensions, D_1, $D_2 \ldots D_N$ and the desired Element indexes x_1, $x_2 \ldots x_n$. Each Element index (x_1, $x_2 \ldots x_n$) is zero based, so the range of x_i is zero to D_i-1, i.e. $x_i = [0, D_i\text{-}1]$. E_S denotes the Element size (E_{Bytes}) of each Element in the Array.

1-Dimensional Array

$$\text{Offset}(\text{column}) = \text{Offset}(x_1) = x_1 E_s$$

2-Dimensional Array

$$\text{Offset}(\text{row}, \text{column}) = \text{Offset}(x_1, x_2) = (x_1 D_2 + x_2) E_s$$

3-Dimensional Array

$$\text{Offset}(\text{plane}, \text{row}, \text{column}) = \text{Offset}(x_1, x_2, x_3) = (x_1 D_2 D_3 + x_2 D_3 + x_3) E_s$$

N-Dimensional Array

$$\text{Offset}(x_1, \ldots, \text{plane}, \text{row}, \text{column}) = \text{Offset}(x_1, x_2, \ldots, x_N) = \left(\sum_{i=1}^{N} x_i \left(\prod_{j=i+1}^{N} D_j \right) \right) E_s$$

Appendix C Normalizing Element Lengths

The Natural MDAP requires all Elements within the Array to be the same length. By either reducing or inflating selected Elements to the desired length, the size of the elements normalizes to the same length. Reducing data length is possible depending on the application and assuming no loss in data integrity. For example, if the data Elements are null-padded strings then reduce the length by removing trailing null characters, if they do not provide any meaning.

Elements inflated or upsized to match the lengths of larger Elements need to follow a consistent method of inflating. Each type of data will require a different method as indicated in Table 2.

Table 2: Data Inflation Methods

Type	Inflation Method
Signed Integer	Add sign extended bytes by adding additional most significant bytes. If the most significant bit of the original value is zero (0), then additional bytes will have all bits zero (0x00). If the most significant bit of the original value is one (1), then additional bytes will have all bits set (0xFF).
Unsigned Integer	Add additional most significant bytes with zero filled bytes (0x00)
32-bit Float	Type Cast a 32-bit floating point value into a 64-bit floating point value
Strings	Pad the end of the string with null (0x00) characters

To deflate the data back to the original size, if known, perform the Data Inflation method in reverse.

Appendix D APA (Array Processing Algorithm)

An Array Processing Algorithm enables packing or other processing on Array Elements. Table 3 lists the available Array Processing Algorithms.

Table 3: Array Processing Algorithms

APA Value	Algorithm	Reference	Format Type	Number of Parameters
0x00	Reserved – Do not use	N/A	N/A	0
0x01	Natural Format – No Element Processing or Encoding	N/A	Natural	0
0x02	MISB ST 1201 Floating Point to Integer Mapping	Appendix D.1	Element-Processed	2
0x03	Boolean Array Encoding	Appendix D.2	Compact	0
0x04	Unsigned Integer Encoding	Appendix D.3	Compact	1
0x05	Run-Length Encoding	Appendix D.4	Compact	1

Appendix D.1 MISB ST 1201 Element Processing

In formatting floating point Elements into KLV, MISB ST 1201 [3] reduces the number of bytes per Element by mapping floating point values to integers. There are two methods specified in ST 1201 for mapping the values: IMAPA and IMAPB. IMAPA parameters are the minimum, maximum and precision values to compute the length of a mapped integer value; IMAPB parameters are the minimum, maximum and a pre-computed length value. Table 4 describes the minimum and maximum values per ST 1201.

Table 4: MISB ST 1201 Minimum/Maximum Values

Name	Description
ST 1201 Minimum Value	The Minimum Value used for mapping and reverse mapping any value in the Array to its corresponding packed value - see MISB ST 1201 for further details.
ST 1201 Maximum Value	The Maximum Value used for mapping and reverse mapping any value in the Array to its corresponding packed value - see MISB ST 1201 for further details.

Requirement	
ST 1303-17	When using APA 0x02, the APAS value shall contain the Minimum and Maximum values as defined in MISB ST 1303 Table 4: MISB ST 1201 Minimum/Maximum Values.

The value of E_{Bytes} along with the Minimum and Maximum values form the IMAPB mapping parameters for each value in the Array: IMAPB(Minimum, Maximum, E_{Bytes}). Because the MDAP already specifies the length of an Element in the Array, this standard uses the IMAPB method of MISB ST 1201.

Requirement	
ST 1303-19	The IMAPB Minimum and Maximum parameters shall both be the same size: either both 32-bit or both 64-bit IEEE [5] floating point numbers.

The combined length of the Minimum and Maximum parameters is computed from the total length of the Pack minus the length of the other parameters (N_{Dim}, Dim_1, …, Dim_N, E_{Bytes} APA and Array of Elements), as shown below.

MinMax Length = Pack Length - ($L(N_{Dim}) + L(Dim_1) + \ldots + L(Dim_N) + L(E_{Bytes}) + L(APA) + L(\text{Array of Elements})$)

Where:

MinMax Length = L(Minimum) + L(Maximum)

L(**x**) is the length of value **x** in bytes.

MinMax Length will be either 8 bytes or 16 bytes. When the MinMax length is 8 bytes then the Minimum and Maximum values are single precision, so the first four bytes are the Minimum parameter and the last four bytes are the Maximum parameter. When the MinMax length is 16 bytes the Minimum and Maximum values are double precision, so the first 8 bytes are the Minimum parameter and the last 8 bytes are the Maximum parameter.

Appendix D.2 Boolean Array Encoding

A multi-dimensional array of Boolean values compacts into a series of bytes with each bit in the bytes representing one Boolean value. With zero (0) equaling “false” and one (1) equaling “true”, the multi-dimensional Boolean array is serialized into a single dimensional linear array as described in Section 7.2.5. The resulting byte representation groups each eight bits into a byte for the whole sequence of Boolean values. The final byte has extra zero bits appended if the resulting number of bits is not a multiple of eight. Figure 6 illustrates the serialization, padding, and conversion to byte values for an example 5x4 Binary Array. The resulting value is three bytes for the 20 Boolean values.

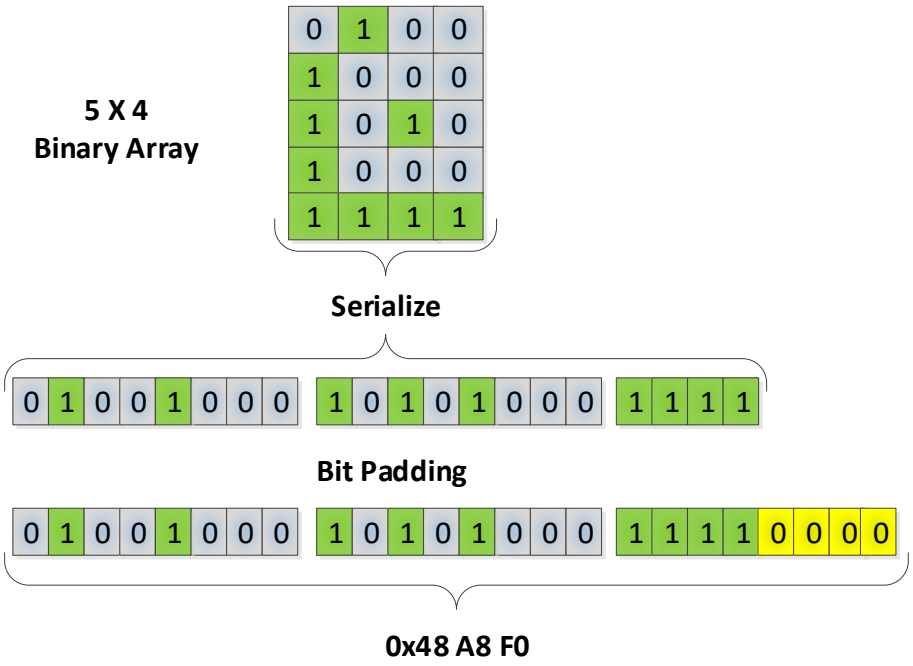

Figure 6: Boolean Array Example

To reverse the process, the product of the dimension values (D_1 .. D_n) determines the number of valid bits converted back to the multidimensional Boolean array.

When using Binary Array coding the E_{Bytes} value is set to one (1).

Appendix D.3 Unsigned Integer Encoding

A multi-dimensional array of unsigned integers (UInt) encodes each Element as a BER-OID. The multi-dimensional UInt array serializes into a single dimensional linear array. Each value of the serialized array is BER-OID encoded and combined into a single block of BER-OID data. Because each value is a variable length Element, extracting individual Elements from the serialized array is not possible until decoding the whole array back into UInt values. Figure 7 illustrates the encoding of a 3x3 array of UInts into a binary block of bytes using BER-OID for each value. The example shows serialization of the 3x3 array into a linear array of nine values, where each value is BER-OID encoded producing values with varying length depending on their numeric value. The final data block contains 12 bytes which is less than the fixed length method which uses 18 bytes (each value requires two bytes).

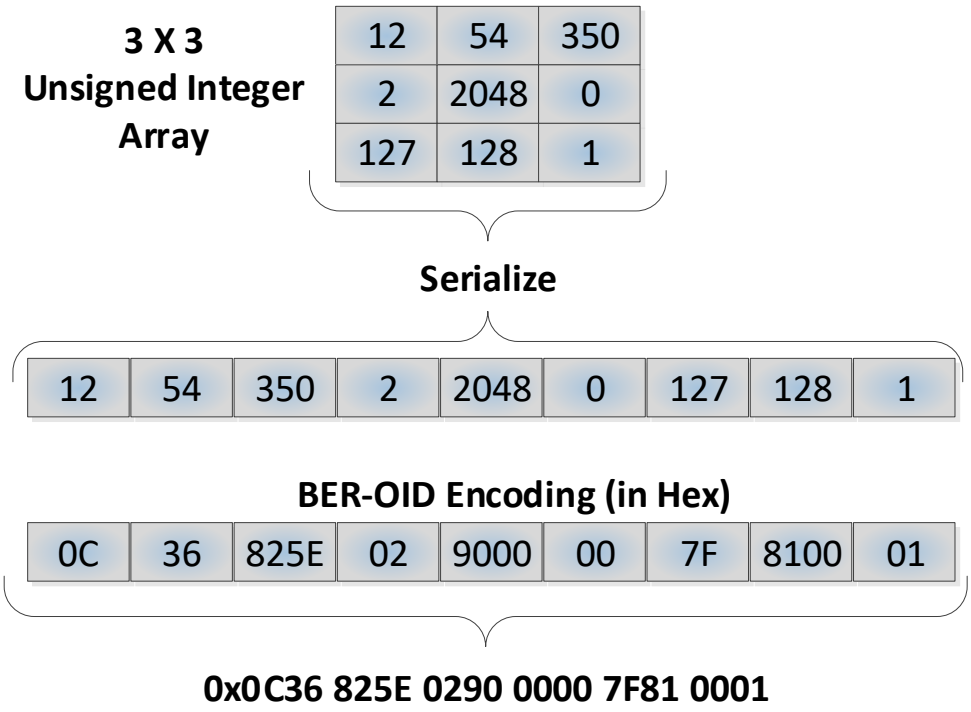

Figure 7: Example Unsigned Integer Array Encoding

Reverse the process to decode the BER-OID block of data to a multi-dimensional array. First, BER-OID decode the values into a one-dimensional array of UInt values, then convert the one-dimensional array back to a multi-dimensional array using the dimension items from MDAP.

An optimization for an array with large numbers is to subtract a bias value from all Elements in the array. The bias value is the smallest value in the array. For example, a one-dimensional array with the values [130, 170, 155, 143, 190] has a minimum value of 130, which when subtracted from each Element produces a "biased" array with the values [0, 40, 25, 13, 60]. In the original array each value requires two bytes for BER-OID encoding for a total of 10 bytes; in the "biased" array each value requires only one byte for a total of 5 bytes plus two bytes to encode the bias.

When using Unsigned Integer coding the APA item is set to four (4) and the E_{Bytes} item is set to one (1). The APAS item contains the bias as a BER-OID value.

Figure 8 illustrates the complete MDAP and its length from the example in Figure 7. The Length of 18 (bytes) is the total length of the Multidimensional Array Pack "Value". The array is a 2-dimensional array (N_{Dim} is 2) with 3 rows (Dim_1) and 3 columns (Dim_2). The E_{Bytes} value is set to one (1) because the APA is specifying variable length Elements in the array – this value is a placeholder. The APA method is set to 4, from Table 3, indicating an Unsigned Array Encoding. The APAS value is set to the bias (or minimum value in the array), which in this illustration is zero. The Array of Elements consists of the nine BER-OID array Elements encoded into 12 bytes.

Length	Multidimensional Array Truncation Pack "Value"						
	N_{Dim}	Dim_1	Dim_2	E_{Bytes}	APA	APAS	Array of Elements
18 (BER)	2 (BER-OID)	3 (BER-OID)	3 (BER-OID)	1 (BER-OID)	4 (BER-OID)	0 (BER-OID)	0x0C36 825E 0290 0000 7F81 0001

Figure 8: Unsigned Array Example 1

Figure 9 illustrates the complete MDAP and its length from the biased length example discussed above. The Length of 11 (bytes) is the total length of the Multidimensional Array Pack "Value". The array is a 1-dimensional array (N_{Dim} is 1) with 5 columns (Dim_1). The E_{Bytes} value is set to one (1) because the APA is specifying variable length Elements in the array – this value is a placeholder. The APA method is set to 4, from Table 3, indicating an Unsigned Array Encoding. The APAS value is set to the bias (or minimum value in the array), which in this case is 130 or 0x8102 when BER-OID encoded. The Array of Elements consists of the five BER-OID array Elements.

Length	Multidimensional Array Truncation Pack "Value"					
	N_{Dim}	Dim_1	E_{Bytes}	APA	APAS	Array of Elements
11 (BER)	1 (BER-OID)	5 (BER-OID)	1 (BER-OID)	4 (BER-OID)	130 (0x8102) (BER-OID)	0x0028 190D 3C

Figure 9: Unsigned Array Example 2

Appendix D.4 Run-Length Encoding

Run-Length Encoding (RLE) an array provides a method for reducing the bandwidth for arrays which have contiguous blocks of a constant value. Instead of including each value of an array in the MDAP, an RLE array only lists starting coordinates and the lengths of a constant value. Figure 10 illustrates a 10x10 array of values with row/column numbers on the left and top of the array, respectively. This array has contiguous sections, or "patches" of the same numbers (highlighted in blue, yellow, or green), in blocks throughout the array. This blocking property enables reducing the bandwidth by using Run-Length Encoding which is a list of triplets that includes a value, a starting coordinate, and a run-length. For example, [-1424, (0,3), (10,2)] describes the array values shaded in green with the value -1424, starting in row/column (0,3), that extends for 10 rows and 2 columns (10, 2).

	0	1	2	3	4	5	6	7	8	9
0	1656	1656	1656	-1424	-1424	0	0	0	0	0
1	1656	1656	1656	-1424	-1424	0	0	0	0	0
2	1656	1656	1656	-1424	-1424	0	0	0	0	0
3	1656	1656	1656	-1424	-1424	0	0	0	0	0
4	-1015	-1015	-1015	-1424	-1424	978	978	978	978	978
5	-1015	-1015	-1015	-1424	-1424	978	978	978	978	978
6	-1015	-1015	-1015	-1424	-1424	978	978	978	978	978
7	-1015	-1015	-1015	-1424	-1424	1260	1260	1260	1260	1260
8	-1015	-1015	-1015	-1424	-1424	1260	1260	1260	1260	1260
9	-1015	-1015	-1015	-1424	-1424	1260	1260	1260	1260	1260

Figure 10: Contiguous Blocks in Array for Run-Length Encoding Example

Table 5 provides the complete list of "patches" in the previous example. The Patch Number column is a reference for describing the patches and is not a part of the encoding process. The Value, Starting Location and Run Length were previously defined. The Area Size is the number of Elements the patch includes.

Table 5: List of Patches in Run-Length Encoding Example

Patch Number	Value	Starting Location	Run Length	Area Size
1	1656	(0,0)	(4,3)	12
2	-1424	(0,3)	(10,2)	20
3	0	(0,5)	(4,5)	20
4	-1015	(4,0)	(6,3)	18
5	978	(4,5)	(3,5)	15
6	1260	(7,5)	(3,5)	15

An optimization is to define a "background" value of the array which states the default value for the array Elements (discussed below), then patches over-ride the default value with their patch values. For example, a default value in this example could be the value of patch 2, then patches 1, 3, 4, 5, and 6 define their regions and values. A distributed value (i.e., non-contiguous) in an array or distributed patches (with the same value) in the array are good candidates for the default value.

To describe the contents of the example array using RLE requires: two bytes for the Value, four bytes for the Starting Location, and four bytes for the Run Length for each of the six patches, resulting in 6 x (2 + 4 + 4) = 60 bytes. To reduce this size further use BER-OID values for the

Starting Location and Lengths: 6 x (2 + 2 + 2) = 36 bytes. Without using RLE this 10 x 10 array is 100 x 2 = 200 bytes, therefore, in this example, the bandwidth reduction is 36/200 or 82%.

The RLE method works for any array dimensionality; the Starting Location and Run Length dimensionality matches the dimensionality of the array. As each case is different RLE may not be the most efficient method for the array data. The MISB recommends assessing, at run-time, the best method for transmitting an array.

When using the RLE data format, the Array of Elements is a list of patches with each patch containing the Value, Coordinates (one coordinate for each dimension), and Run Lengths (one Run Length for each dimension). The invoking document defines the numeric type of the value. A single APAS value defines the default value to use for Elements not a part of Patches. The E_{Byte} items states the number of bytes for the default value and each Value in the Patch description. For example, if the E_{byte} value is four (4), then the default value and all the Patch's Values will be 32 bits. Figure 11 illustrates the Array of Elements and the format of each Patch description.

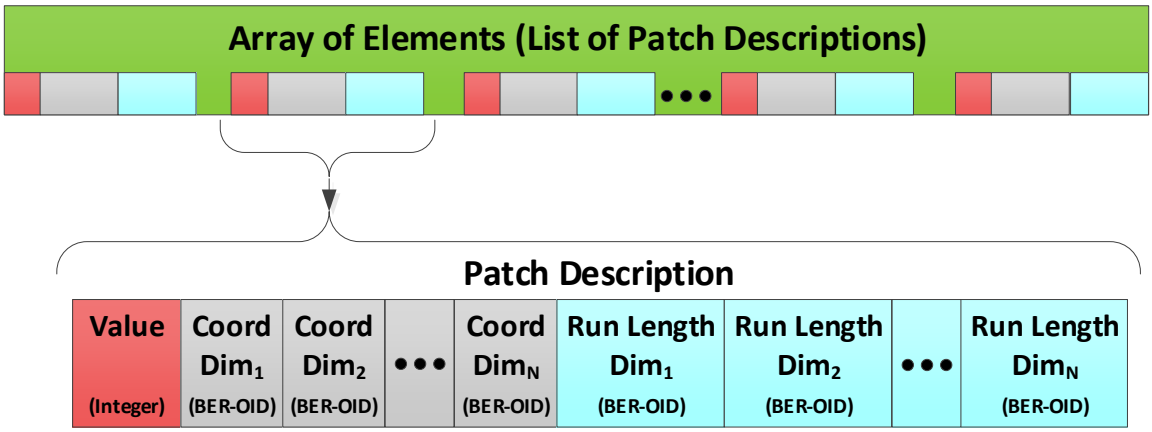

Figure 11: Run-Length Encoding Format

Figure 12 illustrates the complete MDAP for the example in Table 5. The items in red are the values from the example. The Length of 37 (bytes) is the total length of the Multidimensional Array Pack value. The array is a 2-dimensional array (N_{Dim} is 2) with 10 rows (Dim_1) and 10 columns (Dim_2). The E_{Bytes} value is set to two (2) because the APAS value and each Patch Value is two bytes. The APA method is set to 5, from Table 3, indicating a Run-Length Encoded array. The APAS value is set to the default value for the array, which in this case is -1424. The Array of Elements consists of five Patch Descriptions, each containing a Value, two Coordinates, and two Run Lengths.

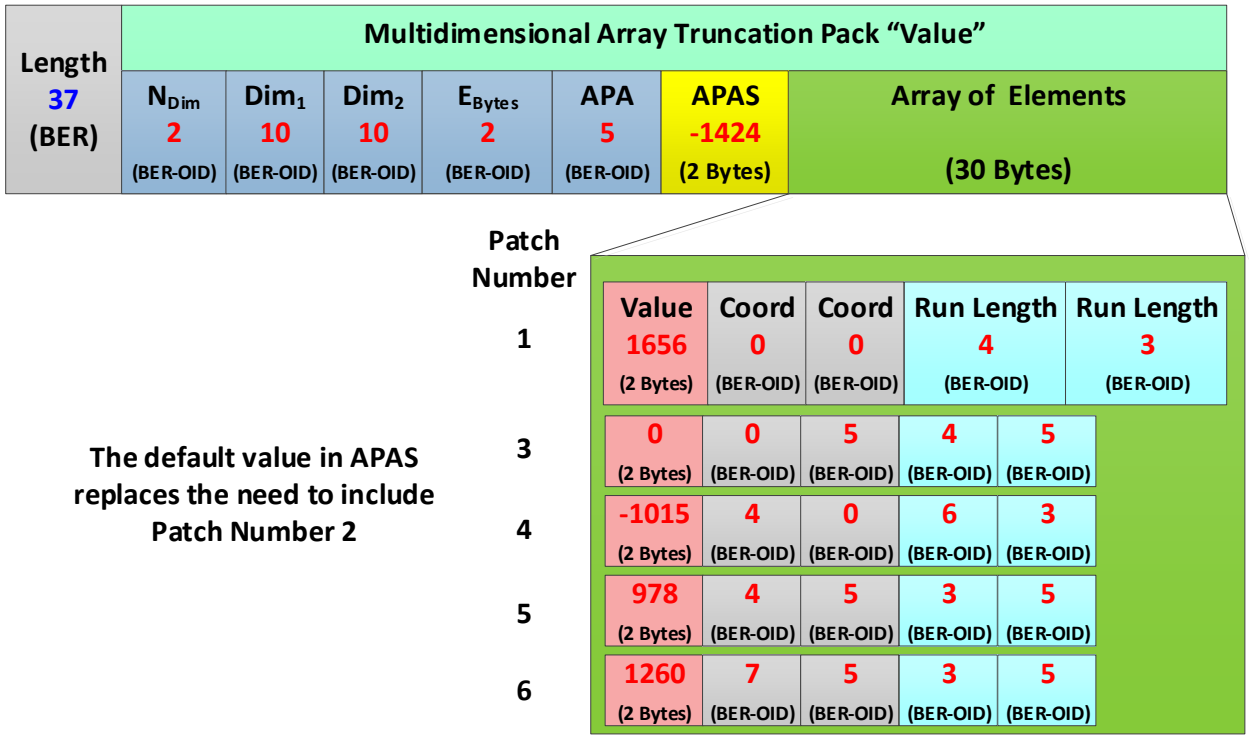

Figure 12: Integer Run-Length Encoding Example

Converting the above information into a single hex value result in:

0x2502 0A0A 0205 FA70 0678 0000 0403 0000 0005 0405 FC09 0400 0603 03D2 0405 0305 04EC 0705 0305

Another optimization is to allow patches to overlap. This means the order of the patches in the list is important since the last patch may "overwrite" values from previous patches. Figure 13 illustrates the same pattern as the example in Figure 10 but where some patches contain the same value. The Figure 10 example resulted in six patches, while this example only requires four overlapping patches: [27, (0,0), (10,3)], [0, (0,5), (10,5)], [13,(4,0), (3,10)], [275,(0,3), (10,2)].

	0	1	2	3	4	5	6	7	8	9
0	27	27	27	275	275	0	0	0	0	0
1	27	27	27	275	275	0	0	0	0	0
2	27	27	27	275	275	0	0	0	0	0
3	27	27	27	275	275	0	0	0	0	0
4	13	13	13	275	275	13	13	13	13	13
5	13	13	13	275	275	13	13	13	13	13
6	13	13	13	275	275	13	13	13	13	13
7	27	27	27	275	275	0	0	0	0	0
8	27	27	27	275	275	0	0	0	0	0
9	27	27	27	275	275	0	0	0	0	0

Figure 13: Overlapping Patches Example

Figure 14 illustrates the complete MDAP for the example in Figure 10. The items in red are the values from the example. The Length of 25 (bytes) is the total length of the Multidimensional Array Pack value. The array is a 2-dimensional array (N_{Dim} is 2) with 10 rows (Dim_1) and 10 columns (Dim_2). The E_{Bytes} value is set to two because the APAS and Path Values are two bytes (needed to support a value of 275). The APA method is set to 5, from Table 3, indicating a Run-Length Encoded array. The APAS value is set to the default value for the array, which in this case is zero (0). The Array of Elements consists of three Patch Descriptions, each containing a Value (two bytes), two Coordinates, and two Run Lengths.

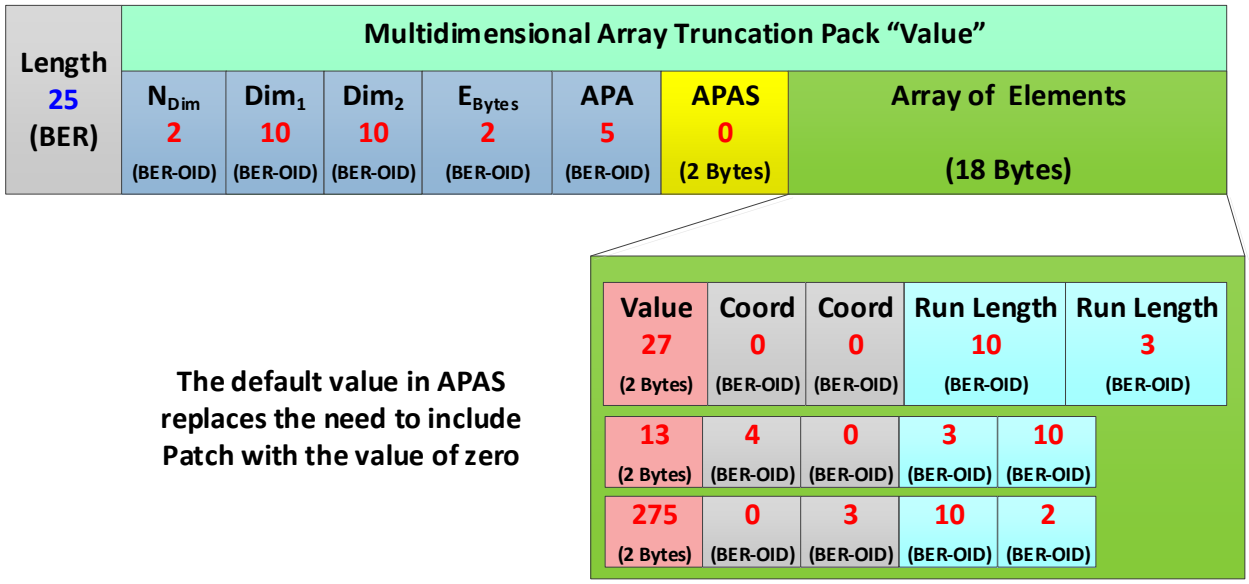

Figure 14: Overlapping Patches Pack Example

Converting the above information into a single hex value result in:

0x1902 0A0A 0205 0000 001B 0000 0A03 000D 0400 030A 0113 0003 0A02